(2)

SHORT ENCODINGS OF EVOLVING STRUCTURES

Daniel D. Sleator
Robert E. Tarjan
William P. Thurston

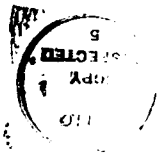
CS-TR-265-90

April 1990

Princeton Univ
Dept. of Computer Science

Short Encodings of Evolving Structures

Daniel D. Sleator [1]
Robert E. Tarjan [2]
William P. Thurston [3]



April 26, 1990

Abstract

A derivation in a transformational system such as a graph grammar may be redundant in the sense that the exact order of the transformations may not affect the final outcome; all that matters is that each transformation, when applied, is applied to the correct substructure. By taking advantage of this redundancy, we are able to develop an efficient encoding scheme for such derivations. This encoding scheme has a number of diverse applications. It can be used in efficient enumeration of combinatorial objects or for compact representation of program and data structure transformations. It can also be used to derive lower bounds on lengths of derivations. We show for example that $\Omega(n \log n)$ applications of the associative and commutative laws are required in the worst case to transform an n-variable expression over a binary associative, commutative operation into some other equivalent expression. Similarly, we show that $\Omega(n \log n)$ "diagonal flips" are required in the worst case to transform one n-vertex numbered triangulated planar graph into some other one. Both of these lower bounds have matching upper bounds. An $O(n \log n)$ upper bound for associative, commutative operations was known previously, whereas we obtain here an $O(n \log n)$ upper bound for diagonal flips.

---

[1]Computer Science Department, Carnegie Mellon University, Pittsburgh, PA 15213. Research supported in part by the National Science Foundation under grant CCR-8658139, by the Defense Advanced Research Projects Agency, monitored by the Space and Naval Warfare Systems Command under contract N00039-87-C-0251, and by DIMACS, a National Science Foundation Science and Technology Center, Grant No. NSF-STC88-09648.

[2]Computer Science Department, Princeton University, Princeton, NJ 08544 and NEC Research Institute, Princeton, NJ 08540. Research at Princeton University partially supported by the National Science Foundation, Grant No. DCR-8605962, the Office of Naval Research, Contract No. N00014-87-K-0467, and by DIMACS.

[3]Mathematics Department, Princeton University, Princeton, NJ 08544. Research partially supported by the National Science Foundation, Grant No. DMS-8806067.01.

1. Introduction

The object of this paper is to study succinct representations of derivations in transformational systems. To model transformational systems, we use *graph grammars* [2]. Roughly speaking[1], a graph grammar consists of a finite set of productions $\{L_i \rightarrow R_i\}$. Each production $L_i \rightarrow R_i$ consists of a connected graph L_i, called the *left side* of the production, and a graph R_i, called the *right side* of the production. A production $L_i \rightarrow R_i$ is *applicable* to a graph G if G contains a subgraph isomorphic to L_i. The production is applied to G by replacing an occurrence of L_i in G by a copy of R_i. (There may be more than one way of applying a production to G, since G may contain more than one copy of the left side.) A *derivation* is a sequence of graphs $G = G_0, G_1, G_2, \cdots, G_m = G'$ such that each G_i is obtained from G_{i-1} by applying one production once. The derivation *transforms* graph G into graph G'. A particular application of a production during a derivation is called an *action*.

Let Γ be a fixed graph grammar, and let G be a fixed starting graph of size n. Consider the collection $R(G, \Gamma, m)$ of all graphs obtainable from G by derivations of length m or less. Our main result is an efficient method of encoding any graph in $R(G, \Gamma, m)$. To encode any such graph uses $O(m + \log \binom{n}{m})$ bits. (In most cases of interest, $m > n$, and the second term in this bound is zero.) This bound is an improvement by a logarithmic factor over the obvious bound of $O(m \log s)$, where $s \geq n$ is the size of the largest graph occurring in the derivation [9]. This logarithmic improvement is crucial in obtaining the tight lower bounds discussed below.

Our *encoding* represents an equivalence class of derivations obtained by permuting commutative applications of the productions. The efficiency of such an encoding arises from the fact that there may be many derivations equivalent to any given one, a fact that follows from the localized nature of applications of the production rules. For simplicity we formulate our result in the setting of labeled directed graphs; it holds for more general combinatorial structures such as hypergraphs and simplicial complexes, however.

Our result has a number of general and specific applications, both theoretical and practical. Our main theoretical application is in demonstrating the existence of pairs of graphs that are far apart, in the sense that any derivation of one graph of the pair from the other must take many actions. If $N(G)$ is a lower bound on the number of graphs derivable from a graph G of size n, then there is a graph G' such that any derivation of G' from G has length $\Omega(\log N(G) - n)$. This is because our encoding scheme implies that the number of graphs derivable from G' by derivations of length m or less is at most c^{n+m}, for some constant c that depends only on the grammar and not on m and n.

Our first application involves transformations of arithmetic expressions. Consider the collection of fully parenthesized expressions of n variables over an associative, commutative binary operation. A *move* consists of applying either the commutative law (exchanging two subexpressions that are combined by the operation) or the associative law (erasing a pair of matching parentheses to put three expressions at the same level, and adding a new pair of parentheses to alternatively regroup this triple). We show that given any n-variable

[1]Section 2 gives a precise definition of the form of graph grammar that we use.

expression E, there is an equivalent expression whose distance from E in this metric is $\Omega(n \log n)$. This solves an open problem of Culik and Wood [1], who obtained a matching upper bound. Thus the worst-case distance between two equivalent expressions is $\Theta(n \log n)$. This contrasts with the corresponding bound of $2n - O(1)$ if commutativity is not allowed [6].

As a second application of our lower bound, we consider the collection of numbered triangulations of the plane, transformed by the "flip" operation. This operation removes an edge, thereby creating a quadrilateral face, and replaces it with the other diagonal of the face. A flip is only allowed if it does not create a multiple edge. Our encoding method proves that there exist pairs of n-vertex triangulations that are $\Omega(n \log n)$ flips apart. We show, furthermore, that this bound is tight by giving a method for converting any n-vertex triangulation into any other in $O(n \log n)$ flips. This improves the previous $O(n^2)$ upper bound of Wagner [8].

We envision several other applications of our technique. First, it can be used to efficiently encode graphs or other combinatorial structures that are close to a given one (in the sense of being obtainable by a small number of transformations). Such encodings may be useful in situations that require the representation of multiple versions of a structure, as in program transformation systems and other applications of *persistent* data structures [3]. Second, it provides a way to enumerate graphs of various kinds that are generated by graph grammars or other such transformational systems. By enumerating our encodings rather than enumerating sequences of productions, all of the desired graphs will be generated, but with far fewer redundant copies of isomorphic graphs.

The remainder of this paper consists of five sections. In Section 2 we give a precise formulation of graph grammars and graph grammar derivations, describe our encoding scheme for derivations, and use this to prove upper bounds on the number of graphs obtainable by short derivations. Section 3 gives several refinements and improvements of our method. Sections 4 and 5 show how the encoding scheme applies to prove our lower bound results for expressions and plane triangulations. Section 6 contains our upper bound on the distance between plane triangulations; it is independent of the rest of the paper.

2. Encoding Graph Derivations

We shall be concerned with graphs that are undirected and of degree at most b (a fixed constant independent of n). Each end of each edge is labeled with an integer called an *edge-end label*. The edge-end labels incident on a vertex are distinct and between 1 and b inclusive. It will be useful to be able to refer to *half* of an edge. Each such *half-edge* has one end vertex from the original edge, and one edge-end label. We allow the graph to have multiple edges between the same pair of vertices, and even to have self-loops. (It is easy to modify our construction to disallow such structures, although doing so would only weaken our lower bounds.)

A *graph grammar* (usually denoted by Γ) is a finite set of productions $\{L_i \rightarrow_i R_i | i = 1, 2, \ldots\}$. The ith production is comprised of three parts: L_i, the *left side* of the production;

R_i, *the right side* of the production; and $\rightarrow_i$, the *correspondence* of the production. The three parts of a production have the following characteristics:

- L_i: This is a connected, undirected, edge-end labeled graph, with degree bounded by b. Strictly speaking, L_i is not a graph, because it has a set of half-edges that have only one end vertex. The one end vertex of each half-edge that is attached to a vertex of L_i has an edge-end label.
- R_i: This is also a graph with edge-end labels, and half-edges, of which it has the same number as L_i.
- $\rightarrow_i$: This is a one-to-one map between the half-edges of L_i and those of R_i.

The production $L_i \rightarrow_i R_i$ applies to a graph G if G contains a set of vertices S such that $G(S)$ (the subgraph induced by S in G) is isomorphic (including edge-end labels) to L_i. The induced subgraph $G(S)$ is most simply defined by retaining half of every edge incident to a vertex in S. The half-edges of $G(S)$ come from the edges of G with one end-point in S and one not in S. The production is applied by replacing this occurrence of L_i in G by R_i, where each half-edge of R_i is attached to a half-edge of $G - G(S)$ just as the corresponding half-edge of L_i was attached. Sections 3 and 4 give examples of specific graph grammars.

Each vertex occurring in an L_i has a unique *position number* from the set $\{1, 2, \ldots, c\}$, where c is the total number of vertices in all left sides. The position numbers will be used to uniquely specify a vertex in a production. The vertices of each R_i are also numbered $1, 2, \ldots$ within each production. These numbers are the *right position numbers*.

A *derivation* is a sequence of graphs $G = G_0, G_1, \ldots, G_m = G'$ such that each G_i is obtained from G_{i-1} by applying one production once. An *action* is a particular application of a production during the derivation. The derivation *transforms* G into G'. The *length* of such a derivation is m, the number of actions in it.

We shall construct a pair of functions $\text{ENCODE}_{G,\Gamma}$ and $\text{DECODE}_{G,\Gamma}$. The function $\text{ENCODE}_{G,\Gamma}$ takes a derivation D that transforms G into some other graph G', and returns a string of symbols from the alphabet $\Sigma = \{0, 1, 2, \ldots, c\}$ of length $n + r \cdot m$. Here n is the number of vertices of G, m is the length of the derivation D, c is the number of vertices in left sides of Γ, and r is the number of vertices in the largest right side of Γ. This sequence is called the *encoding* of the derivation. The function $\text{DECODE}_{G,\Gamma}$ takes as input such an encoding and returns the graph G'. That is,

$$\text{DECODE}_{G,\Gamma}(\text{ENCODE}_{G,\Gamma}(D)) = G'.$$

For our purposes it is useful to think of the process of applying a production as *destroying* vertices (the ones that are matched to the vertices of L_i) and *creating* new and different ones (the ones introduced by R_i). The actions of a derivation D of length m are numbered $1, 2, \ldots, m$ in the order in which they occur. Each vertex that is created during the derivation can be identified uniquely by specifying the number of the action that created it and the

position number of the vertex in R_i that created it. This is the *name* of the vertex. The *required vertices* of an action are the vertices that are destroyed by it. An action is said to be *ready* at some time during a derivation if all of its required vertices exist at that time. Readiness implies that the entire copy of L_i that is to be replaced (including all of its edges and half-edges) is present as well.

Lemma 1 *Consider a derivation D that transforms G into G'. If the actions of D are reordered in any way so that each production is ready when it is applied, then the new derivation also transforms G into G'.*

Proof. By induction it is sufficient to prove that if a_t and a_{t+1} (two consecutive actions of D) are such that none of the required vertices of action a_{t+1} is created by a_t, then if these actions are swapped the resulting derivation also transforms G into G'. Since either order is allowed, we know that those vertices created by a_t are not used by a_{t+1} and those created by a_{t+1} are not used by a_t. It follows that the actions commute, since they do not involve any of the same vertices. □

We can now give an explicit algorithm for computing the encoding of a derivation D. First, the actions of D are numbered, the vertices of the derivation are named, and the required vertices of each action are computed.

Our encoding algorithm will assign to each vertex of the derivation a unique *number*. First, the vertices of G are numbered $\{1, 2, \ldots n\}$ in an arbitrary order. (The same ordering must be used by the decoding procedure described below.) The remaining vertices are numbered in conjunction with the construction of a *canonical* derivation D', which is a reordering of the actions of D.

The actions of the canonical derivation D' are computed one by one. At any time it is easy to determine which actions are ready; these are the ones whose required vertices exist. Let q be the ready action that destroys the lowest numbered vertex among all ready actions. This action is the one chosen to be the next action of D'. This action is applied to the graph, and the vertices created by it are now numbered consecutively starting after the largest vertex number used thus far. (If several vertices are created by q, they are numbered in the order of the right position numbers in the right side of the production that created them.)

After computing the canonical derivation D', the algorithm proceeds to compute a *label* for every vertex that occurs in the derivation. The label of a vertex v is zero if v is not destroyed by any action in the derivation. Otherwise it is the position number of the role played by v in the production that destroys it.

The desired encoding is a list of at most $n + r \cdot m$ labels of all the vertices in increasing order by vertex number.

We can now describe the decoding procedure. This algorithm takes the graph G (with vertex numbers that agree with those of the encoding procedure), the grammar Γ, and

the encoding (the list of labels), and determines G'. The procedure works by constructing the canonical derivation D', from which it is easy to get G'. As in most data compression/decompression methods, the decoding algorithm mimics the behavior of the encoding algorithm step-by-step.

The crucial fact about the labels of the vertices existing at any time during the canonical derivation is that from these it is possible to determine exactly which actions were ready at the corresponding stage of the encoding process. This is accomplished by the following *matching* procedure:

> If a vertex v has a non-zero label, the label determines i, the production that eventually destroys v, and also the role v plays in this production. For each such vertex, check its neighborhood to see if it is isomorphic to L_i (including edge-end labels). This check is easy to do since we know which vertex of L_i must match v, L_i is connected, and there are edge-end labels to follow. (Recall that the edge-end labels incident to a vertex are disjoint.) If such a subgraph is found, then the labels of these vertices are checked to see if they match the position numbers of the roles that they are supposed to play in the proposed action. If all of these tests are passed, then the action is ready.

Lemma 2 *The matching procedure determines the ready productions that existed at the corresponding stage of the encoding process.*

Proof. If an action is ready, then the matching procedure will certainly find it, because the vertices corresponding to the left side of the ready action will exist and will be labeled in a way consistent with all the conditions checked above.

On the other hand, suppose that the check described above is satisfied starting from some vertex v. Let i be the production indicated by the label of v, and let S be the set of vertices that form the subgraph isomorphic to L_i. We claim that in any continuation of this derivation all vertices of S must be destroyed simultaneously by a single action. Since all these vertices are destroyed by one action, this action must be ready now.

It remains to show that all vertices of S must be destroyed simultaneously. Consider the first action a in some continuation of the derivation that destroys some vertex w of S. Since a is the first action involving the vertices of S, at the moment action a is applied, all of the vertices of S will exist (and have the same labels). From the vertex w the matching algorithm described above will construct the set S. There is no other possible matching pattern involving w. Therefore the action a destroys all the vertices of S simultaneously. □

Now, given that we know the ready productions and the numbering of the vertices of the current graph, it is easy to find q (the next production of D') because it is the ready action that destroys the lowest numbered vertex. This action is applied to the graph. The vertices created by it are numbered sequentially (as in the encoding procedure) and are labeled as specified by the encoding. This step is repeated to determine all of the productions of D'. The process terminates when there are no ready productions.

The following theorem, which bounds the number of graphs obtainable from a given one as a function of the length of a derivation, is a consequence of our encoding scheme.

Theorem 1 *Let G be a graph of n vertices, Γ be a graph grammar, c be the number of vertices in left sides of Γ, and r be the maximum number of vertices in any right side of a production of Γ. Let $R(G,\Gamma,m)$ be the set of graphs obtainable from G by derivations in Γ of length at most m. Then $|R(G,\Gamma,m)| \leq (c+1)^{n+r\cdot m}$.*

Proof. Encode the derivation using the scheme described above. The length of the encoding is at most $n+r\cdot m$ symbols. This encoding can be padded with zeros so that its length is exactly $n + r \cdot m$. (This will not interfere with the decoding process, since it is self-terminating.) The alphabet is of size $c+1$, so the number of such encodings is $(c+1)^{n+r\cdot m}$. Each graph reachable by m or fewer actions is the outcome of applying the decoding procedure to one of these encodings. Therefore the number of such graphs is at most the number of such encodings. □

3. Generalizations and Improvements

This section describes various extensions and improvements to our encoding scheme, most of which will be used later in this paper.

3.1. Encoding short derivations

Our encoding scheme can be modified to make it more efficient when the length of the derivation is short compared to the size of the starting graph. In this case most of the labels of the vertices of the initial graph are zero. The more efficient encoding specifies which vertex labels are non-zero, and only includes labels for these in the vertex label list. Let k be the number of vertices that have non-zero label in the initial labeling of G, and let m, n, r and c be defined as above. Then the size of this encoding (in bits) is:

$$\lceil \log_2 n \rceil + \lceil \log_2 \binom{n}{k} \rceil + (k + mr)\lceil \log_2(c+1) \rceil.$$

The first term is for bits to encode k, and the second term encodes the subset of vertices with non-zero labels.

Theorem 2

$$\log |R(G,\Gamma,m)| = O(\log \binom{n}{m} + m),$$

where $R(G,\Gamma,m)$ is the number of graphs obtainable by derivations of length at most m in grammar Γ starting from a graph G of n vertices. (If $m > n$ then $\log \binom{n}{m}$ is interpreted as zero).

Proof. Theorem 1 shows that $\log |R(G,\Gamma,m)| = O(n+m)$. If $c \cdot m > \frac{1}{2}n$ then $O(n+m) = O(m) = O(\log \binom{n}{m} + m)$.

If $c \cdot m \leq \frac{1}{2}n$ then we use the encoding scheme described above. Since each action causes at most r vertices of G to have non-zero labels we know that

$$k \leq r \cdot m \leq \frac{1}{2}n.$$

It follows that

$$(k+mr)\lceil \log_2(c+1) \rceil = O(m),$$

and that

$$\log_2 \binom{n}{k} \leq \log_2 \binom{n}{rm} \leq \log_2 \binom{n}{m}^r = r \log_2 \binom{n}{m}.$$

Finally, we know that $\log_2 n \leq \log_2 \binom{n}{m}$. The theorem follows from these inequalities and the bound on the number of bits used by the efficient encoding scheme. □

3.2. Leaders and followers

The labels on the set of vertices destroyed by an action contain redundant information. For example, each label of this set has sufficient information to determine which production is the one that destroys all of them. There is a way to eliminate this redundancy and thereby reduce the size of the encoding in most cases.

The new encoding algorithm begins by computing the standard encoding described above. It then applies a map f to each symbol of the encoded string, giving the new encoding. It remains to define the map f.

Let one vertex of each L_i be chosen to be the *leader*, and let all the other vertices be *followers*. For each L_i choose a spanning tree. (This can be done, since each left side is connected.) For each follower vertex v, let $DIR(v)$ be the value of the edge-end label of the v end of the first edge on the path (in the spanning tree) from v to the leader of L_i. (In other words, starting from any vertex in L_i, following the $DIR(\cdot)$ direction repeatedly will lead to the leader.)

The map f is defined as follows ($|\Gamma|$ is the number of productions of Γ, and $v(x)$ is the vertex of a left side with position number x.):

$$f(x) = \begin{cases} 0 & \text{if } x = 0 \\ i & \text{if } v(x) \text{ is the leader of } L_i \\ |\Gamma| + DIR(v(x)) & \text{if } v(x) \text{ is a follower} \end{cases}$$

The decoding algorithm must be modified to accommodate this new encoding. The only difference is in the matching step, which is revised as follows:

For each vertex v that is a leader, check its neighborhood to see if it is isomorphic to L_i. If such an isomorphic subgraph is found, then the labels of these vertices are checked to see if they are all followers, and that if a directed edge is drawn from each follower w in the direction of $DIR(w)$ (which is the label of w minus $|\Gamma|$) then the result is a directed spanning tree rooted at v. If all of these tests are passed, then the action is ready.

We now need to verify that Lemma 2 still holds; that is, that the sets of vertices satisfying the new matching procedure above exactly correspond to the ready actions. The first part of the proof remains easy; any ready action of the original derivation gives rise to a match in the above procedure. On the other hand, a match also indicates that the corresponding action is ready. Let S be the set of matched vertices. Starting from any follower vertex $w \in S$, the entire set S can be constructed uniquely. Similarly, from the leader vertex v of S the set S can be uniquely constructed. This is a sufficient condition to guarantee that all vertices of S are destroyed simultaneously, which (as shown above) is the condition that we need in order to prove that the action is ready.

It may be possible to further reduce the alphabet size by making use of the flexibility that exists in choosing which spanning tree to use on each left side. The number of labels can be reduced from $|\Gamma| + d + 1$ to $|\Gamma| + d' + 1$, where d' is the number of different directions used in the directed spanning trees of the left sides.

The leader-follower technique applies in any situation in which there is a production with more than one vertex on the left side. It may decrease the size of the label alphabet, but it can never increase it. If the technique applies, then it can be used in conjunction with the next technique to further reduce the alphabet size.

3.3. Eliminating the zero label

Suppose that for any graph occurring in a derivation using Γ, there exists a way of labeling it with non-zero labels so that no production is ready. Then the zero label can be eliminated. The encoding procedure must be modified slightly to eliminate the zero labels, while the decoding procedure will remain the same.

Here is how the encoding procedure is revised. First compute the labeling of all the vertices as before. The vertices with zero labels are exactly those that end up in G', the final graph of the derivation. These are called the *terminal* vertices. Compute the labeling of G' with non-zero labels so that no production is ready. For each terminal vertex, replace its label with that terminal label just computed in G'.

It is easy to see that this works by reviewing the proof of Lemma 2. The proof only differs at the point where it is shown that if the labels match the pattern of some left side L_i, then the production i applied to that set of vertices S is ready. The crucial statement is that if this situation occurs, then all the vertices of S must be destroyed simultaneously. This is still true. All of the vertices cannot be terminal ones, since their labels admit the application of a production. The set cannot be comprised of both non-terminals and terminals, because

then the non-terminals would never be allowed to change. Therefore all the vertices of S must be non-terminals, and the previous argument shows that the production is ready.

Notice that in any situation in which the leader-follower technique applies, we can eliminate the zero label. This is done by labeling all the terminal vertices to be followers.

3.4. Tags

It is sometimes useful to carry extra information along during a derivation. (Sections 4 and 5 give examples of this.) To accommodate this, we allow each vertex to have a *tag* associated with it. Each production also supplies an arbitrary function that is used to define the values of the tags of the vertices created in terms of the tags of the vertices destroyed. Because the tags are computed locally (as a function only of tags of the vertices on the left side of the production) the commutativity that we have exploited in constructing our encoding is still present. Therefore our encoding method and theorems apply to tagged graphs without any changes.

4. Expressions over an Associative, Commutative Operation

Let $X = \{x_1, x_2, \ldots, x_n\}$ be a fixed set of variables, let $\oplus$ be a binary operation, and let E_n be the set of fully parenthesized expressions over $\oplus$ in which each variable x_i occurs exactly once. We consider the problem of estimating how many applications of the associative and commutative laws are required to transform any expression in E_n into any other.

To make this problem somewhat more concrete, we restate it as a problem on binary trees. Our binary tree terminology is that of Knuth [5]. Let T_n be the set of full binary trees with n external nodes, numbered $1, 2, \ldots, n$. Any permutation of $1, 2, \ldots, n$ is allowed; thus $|T_n| = n!\binom{2n-2}{n-1}\frac{1}{n} = (n-1)!\binom{2n-2}{n-1}$ [4]. We permit two transformations of a tree $T \in T_n$: a *twist*, in which the left and right subtrees of a specified internal node are exchanged, and a *rotation*, in which an internal node changes places with one of its children while symmetric order in the tree is preserved. (See Figure 1.) The problem is to estimate the minimum number of twists and rotations needed to transform any tree in T_n into any other. We denote this number by R_n.

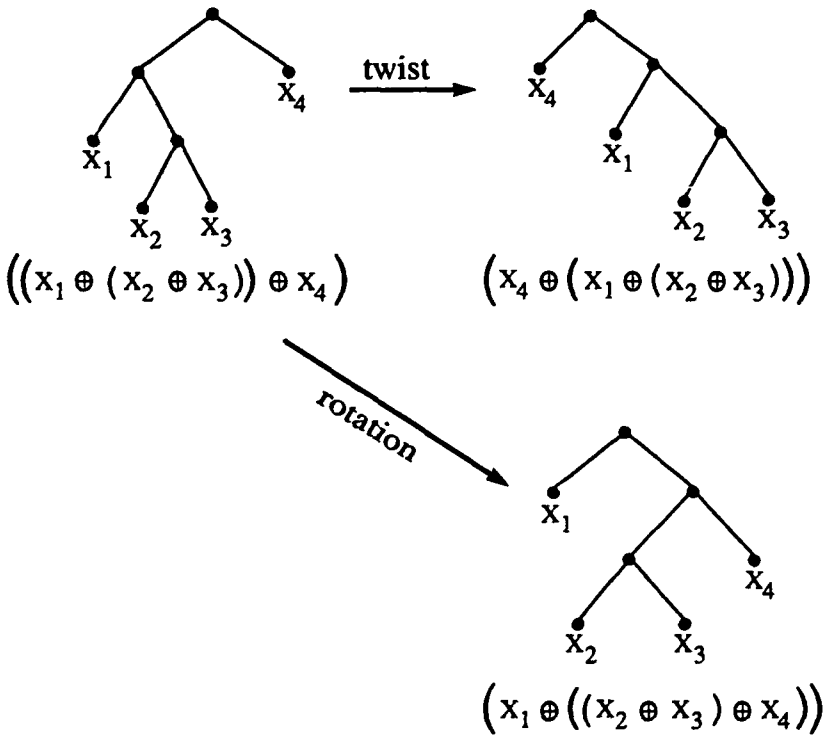


Figure 1: Illustrating a twist and a rotation.

This problem is equivalent to the expression transformation problem. The isomorphism (also shown in Figure 1) between expressions and trees is the standard one – an external node labeled i corresponds to the expression "x_i"; an internal node corresponds to the expression $(E_\ell \oplus E_r)$, where E_ℓ and E_r are the expressions corresponding to the left and right children of the node. A twist corresponds to the application of the commutative law; a rotation, to an application of the associative law.

Culik and Wood [1] derived an $O(n \log n)$ bound on R_n. We shall derive a matching $\Omega(n \log n)$ bound. (Culik and Wood actually worked with a slightly different transformational system, but their result applies to our system, and our result applies to theirs.)

These transformations can be represented as productions in a graph grammar. The graphs we consider differ slightly from the binary trees described above. To transform a tree into the corresponding graph, add an extra node of degree one, called the *superroot*, and connect it to the root of the tree. The edge-end labels of the three edges incident to an internal node are 1, 2, and 3, for the edges connecting the node to its left child, right child, and parent, respectively. (The superroot is the parent of the root.) The $n + 1$ edge-ends that are incident on vertices of degree one are irrelevant, since these will never be involved in any production. The vertices of degree one are tagged with a name that will be carried along during the derivation. The following figure shows an expression tree and the corresponding graph.

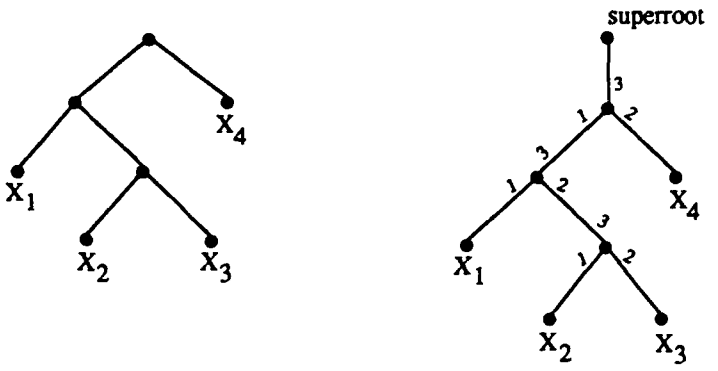


Figure 2: A tree and its corresponding graph.

The grammar to represent this process has three productions, one for a twist, one for a left rotation, and one for a right rotation. These productions are shown in Figure 3.

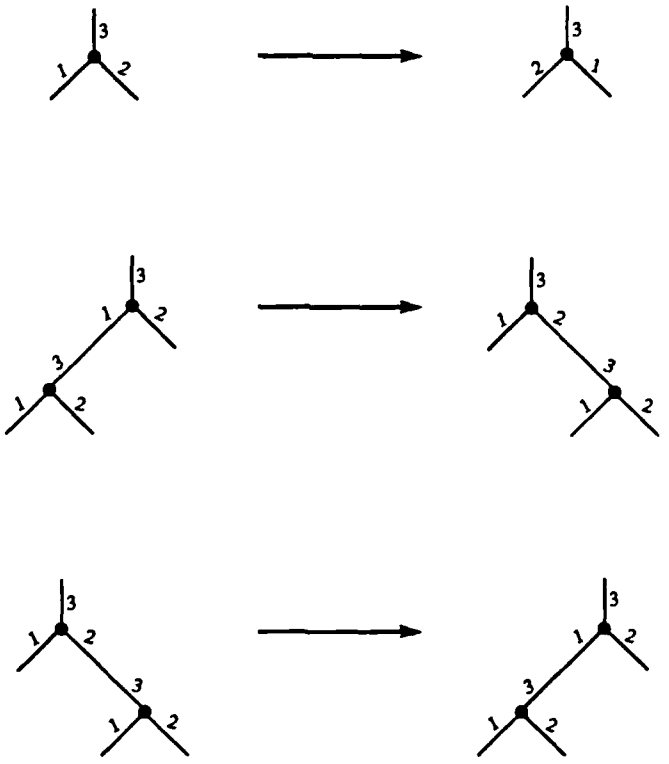


Figure 3: The productions for a twist and rotations. The correspondence between the half edges is obtained by pairing the topmost edges, and walking clockwise simultaneously around the left and right diagrams.

From Theorem 1 it immediately follows that starting from a tree with n external nodes, the number of trees reachable in m or fewer twists and rotations is at most $6^{2n+2m+1}$. The leader-follower technique can be used to prove a tighter bound. By choosing the upper vertex

of the left side of each rotation to be the leader, and the other to be the follower, the label alphabet size is reduced to five. The zero elimination technique now applies. This reduces the alphabet size to four, and the bound to $4^{2n+2m+1}$.

This can be further improved by specializing the encoding and decoding procedures for this application. We do not need to encode the labels for the n leaves or the superroot, because these are not involved in any actions. This improves the bound to 4^{n+2m-1}. The total number of bits needed to encode any tree derivable in m or fewer moves is at most $2n + 4m - 2$.

We summarize this result in the following theorem:

Theorem 3 *For any expression E of n variables:*

1. *The number of different arithmetic expressions obtainable by m applications of the commutative and associative laws starting from E is at most $2^{2n+4m-2}$.*

2. *There exists an expression E' such that the number of operations required to transform E into E' is $\Omega(n \log n)$.*

Proof. Part 1 follows from the prior discussion. Part 2 follows from the fact that there are $(n-1)!\binom{2n-2}{n-1}$ expressions obtainable starting from E. In order to obtain all of them in m moves we must have

$$2^{2n+4m-2} \geq (n-1)!\binom{2n-2}{n-1} = \Omega(n!)$$

$$2n + 4m - 2 = \Omega(n \log n)$$

$$m = \Omega(n \log n)\,.$$

□

5. Numbered Plane Triangulations: A Lower Bound

A *numbered plane triangulation* (henceforth just called a triangulation) is an undirected graph embedded in the plane all of whose faces are triangles and whose vertices are numbered sequentially from 1. We denote by P_n the set of all n-vertex triangulations. A *flip* of an edge in a triangulation is the operation of removing an edge, thereby forming a quadrilateral face, and adding the other diagonal of the face. (See Figure 4.) A flip is allowed only if it

does not introduce a multiple edge.

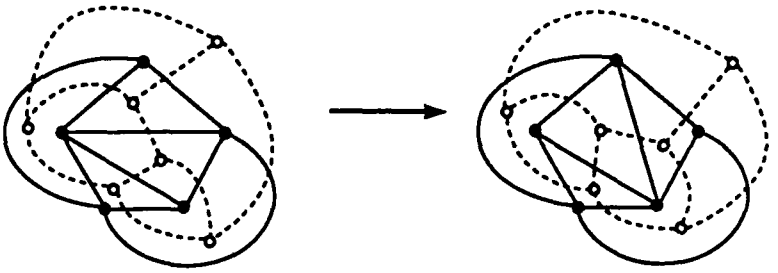


Figure 4: A flip in a triangulated graph, and the corresponding operation in the dual graph.

Let F_n be the minimum number of flips needed to convert any n-vertex triangulation into any other. We wish to estimate F_n. It is not hard to establish that F_n is $O(n^2)$; Wagner [8] gave a construction. We shall show in Section 6 that F_n is $O(n \log n)$; in this section, we use our succinct encoding approach to prove that F_n is $\Omega(n \log n)$.

There is no upper bound on the degree of a vertex in a plane triangulation. Therefore, in order to apply our technique, we shall work in the space of planar graphs that are dual to plane triangulations. In such a graph, every vertex has degree three. (Each vertex of the dual graph (a face in the original graph) maintains as a tag the set of vertex numbers of the vertices in the original graph to which it is incident. These tags along with the dual graph are sufficient to reconstruct the original numbered plane triangulation. This observation is required in order to get a reliable upper bound on the number of reachable numbered plane triangulations.) The edge-end labels of the initial graph are chosen arbitrarily, subject to the constraint that walking one step clockwise around a vertex increases the label by 1 (modulo 3) This ordering of the edge-end labels encodes the embedding of the plane triangulation.

There are several different ways to represent sequences of diagonal flips as derivations in a graph grammar. One way is shown in Figure 5 below.

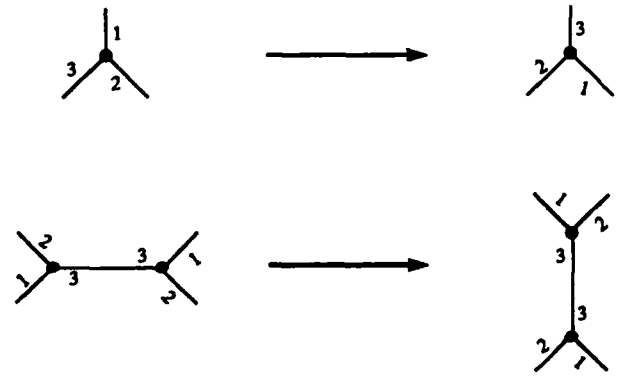


Figure 5: Two productions for representing flip sequences as graph grammar derivations.

This method uses two productions, one for doing the flip, and the other one for preparing the edge-end labels to allow the flip. Each flip in the original derivation may correspond to as many as five actions: two to cycle the edge-end labels on one end, two for the other end, and one for the actual flip. A sequence of m flips turns into a sequence of as many as $5m$ actions. A plane triangulation of n vertices has $2n-4$ faces. Therefore the dual graphs in which the derivations take place have $2n-4$ vertices. The number of vertices in left sides of productions (c) is three, and the number of vertices in the largest right side (r) is two.

We can now apply Theorem 1 to bound the number of n node numbered plane triangulations reachable in m flips by $4^{2n+10m-4}$. This implies that, for any triangulation P, at most $4^{2n+10m-4}$ distinct triangulations can be obtained by doing m or fewer flips. Since P_n contains at least $(n-3)!$ triangulations (there are this many different sorted wheels, see Section 6), there must be at least two triangulations, and indeed many pairs of triangulations, that are $\Omega(n \log n)$ flips apart; that is, $F_n = \Omega(n \log n)$.

The bound on the number of reachable configurations can be tightened significantly by the use of a different graph grammar. This grammar is shown in Figure 6.

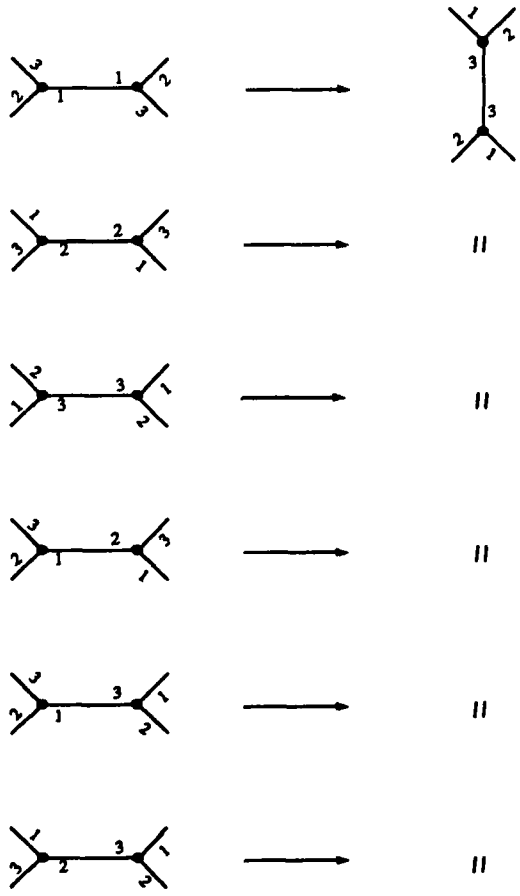


Figure 6: Six productions give a more efficient bound on flip distance.

Because this grammar includes each of the six ways that the ends of the edge to be flipped can be labeled, there is a one-to-one correspondence between diagonal flips in the plane triangulation and applications of one of the productions to the dual graph. Using the leader-follower trick and eliminating the zero label reduces the number of different labels to nine. Each production creates two new labels, so our improved encoding scheme proves that the number of graphs reachable in m moves is at most $9^{2n+2m-4}$.

It turns out that leader vertices are not necessary at all. An encoding without leaders can be made to work by using the convention that a production involving a pair of adjacent vertices is ready if and only if their labels mutually point at each other. To verify that the zero label (indicating a terminal vertex) is not necessary, we need to show that there exists a labeling of any planar graph of degree three with follower labels such that no pair of adjacent vertices point to each other. This can be done as follows. If the graph is a tree, choose a place in the middle of some edge, and make all the vertices point away from this. If the graph has a cycle, choose the labels of the vertices on the cycle to point consistently around it. Now choose a subset of the remaining edges so that these edges plus the cycle form a subgraph with all of the vertices and exactly one cycle. (This is a spanning tree with one extra edge.) The follower label on a non-cycle vertex points toward the cycle along the path in the tree. This gives the required match-free labeling. This argument bounds the number of reachable configurations by $3^{2n+2m-4}$.

The set of configurations reachable in m or fewer flips is not changed if we do not allow a sequence to make a flip then immediately make another flip that cancels it out. This observation means that of the nine possible labelings of the pair of vertices resulting after a move, we can restrict our attention to eight of them. This improves the bound to $3^{2n-4}8^m$.

We summarize the results of this section in the following theorem.

Theorem 4 *For any plane triangulation T of n vertices:*

1. *The number of different plane triangulations obtainable by m or fewer flips starting from T is at most $3^{2n-4}8^m$.*

2. *There exists a plane triangulation T' such that the number of flips required to transform T into T' is $\Omega(n \log n)$.*

6. Numbered Plane Triangulations: An Upper Bound

Theorem 5 *Let G_1 and G_2 be two n-vertex numbered plane triangulations (with no multiple edges). If $n \geq 5$ then there is a sequence of $O(n \log n)$ diagonal flips that transforms G_1 into G_2 in such a way that there are no multiple edges in any intermediate state.*

Proof. We shall show that any such triangulation G can be transformed into a particular canonical form called a *sorted wheel* in $O(n \log n)$ diagonal flips. Using this transformation

we can transform G_1 into the sorted wheel, then transform the sorted wheel into G_2 (using the transformation in reverse).

A *wheel* of $n \geq 5$ vertices is a planar graph that has two special vertices called *hubs*, and $n-2$ other vertices called *rim vertices*. There is an edge from each hub to each rim vertex (these are the *spokes*). There are $n-2$ other edges in the graph, and these form a simple cycle through all of the rim vertices. There is a unique way of embedding the wheel in a sphere.

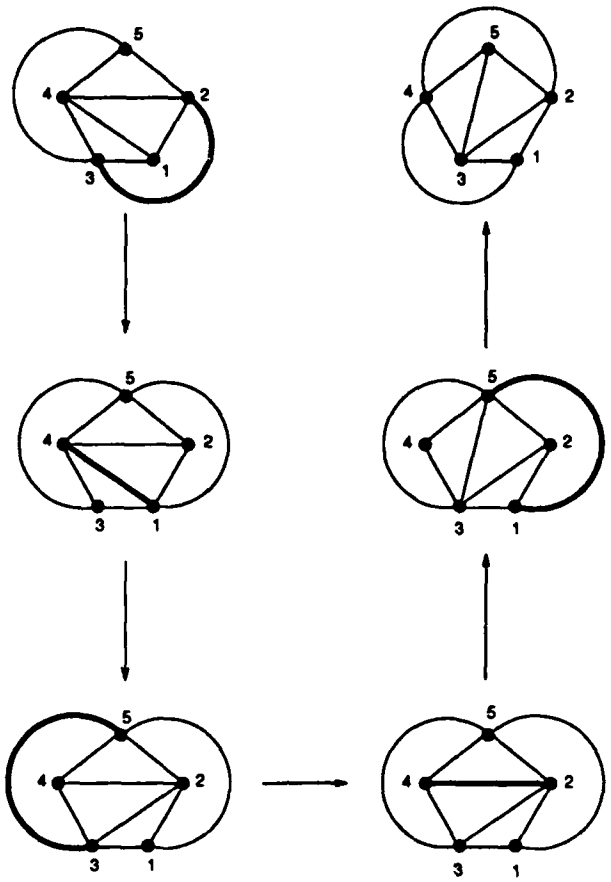
A *sorted wheel* of n vertices is a wheel with labeled vertices embedded in the sphere. The hubs are labeled 1 and n, and the vertices of the rim are labeled $2, 3, \cdots, n-1$ in clockwise order when viewed from hub 1.

We first consider the special case of $n = 5$. Any graph G of five vertices satisfying the hypotheses of the theorem is a wheel. We show this by first applying Euler's formula, which implies that G must have six triangular faces and nine edges, and that the sum of the degrees of the vertices is 18. No vertex can have degree two, because then its two neighbors would be connected by two different edges, which violates the assumption that G has no multiple edges. Furthermore, no vertex can have degree greater than four. It follows that the multiset of the degrees of the vertices is $\{3, 3, 4, 4, 4\}$. The three vertices of degree four must be attached to all the other vertices in the graph. This accounts for all of the edges incident on the vertices of degree three, which therefore must not be neighbors. It follows that the graph is a wheel in which the vertices of degree three are the hubs, and the vertices of degree four are the rim.

We finish the proof for $n = 5$ in two stages. First we show that we can make vertices 1 and 5 the hubs of the wheel. Second we show that if the resulting structure is not the sorted wheel (it must be its mirror image), then it can be transformed into the sorted wheel.

If vertices 1 and 5 are on the rim, then a diagonal flip of the edge between them makes them the two hubs. If 1 is a hub and 5 is on the rim, then we flip the edge between the other two rim vertices creating a configuration where both 1 and 5 are on the rim, which we handle as above. A similar technique suffices if 5 is a hub and 1 is on the rim.

The following diagram shows how the mirror image of the sorted wheel of five vertices is transformed into the sorted wheel by the application of five diagonal flips.



We are now ready to consider the case $n \geq 6$. The transformation of the graph G into a sorted wheel is broken up into three phases: constructing a Hamiltonian circuit, transforming the Hamiltonian circuit into a wheel with hubs 1 and n, and sorting the rim of the wheel. These three steps are described in the following three sections.

6.1. Constructing a Hamiltonian circuit

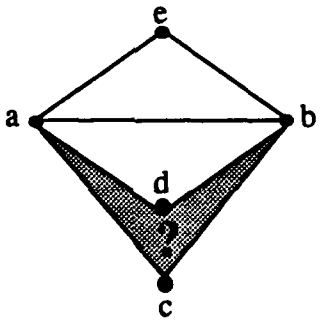
By Tutte's Theorem on planar graphs [7] (and by a theorem of Hassler Whitney [10]), any 4-connected planar graph has a Hamiltonian circuit. The graph G under consideration is 3-connected, since it is planar, triangulated, and has no multiple edges. Unfortunately, it may not be 4-connected. If it is not 4-connected, then it must have a *separating triangle*; that is, a triangle whose removal separates the graph. We shall show how to transform the given graph G into one that has no separating triangles by making $O(n)$ diagonal flips. This will complete our construction of the Hamiltonian circuit.

The graph G is given to us embedded on a sphere. We choose a face arbitrarily and map the embedding on the sphere to an embedding on the plane such that the chosen face is infinite. Each separating triangle of G partitions the faces and remaining vertices of G into two components. The *interior* component is the one not containing the infinite face. Let I_1 be the set of faces interior to a separating triangle T_1, and let I_2 be the set of faces interior to a separating triangle T_2. Either I_1 and I_2 are disjoint, or satisfy $I_1 \subset I_2$ or $I_2 \subset I_1$. It follows

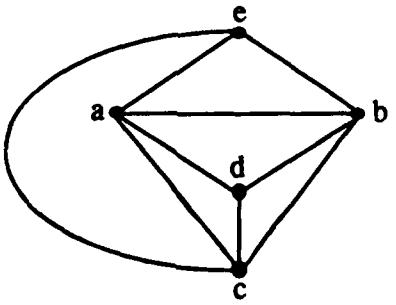
from these relations that there must always be a set of innermost separating triangles, *i.e.*, those that do not contain another separating triangle in their interior.

Our algorithm for eliminating separating triangles works from innermost separating triangles outward. A diagonal flip operation is applied to an edge of one of the innermost separating triangles. The chosen edge will be any one that does not introduce a new separating triangle. We shall prove below that there always exists such an edge. It follows immediately that this algorithm eliminates all of the separating triangles in $O(n)$ diagonal flips, because each flip reduces by at least one the number of edges that are in separating triangles.

It remains to show that there is always an edge of an innermost separating triangle such that if that edge is flipped then no new separating triangle is created. The following case analysis shows this. Consider an innermost separating triangle with vertices a, b, and c. Let d be the vertex inside the triangle such that triangle (a, b, d) is empty. (There must be such a vertex since triangle (a, b, c) is a separating triangle, and there must be something inside of it.) Similarly, there must be a vertex e outside of triangle (a, b, c) such that (a, b, e) is an empty triangle. The figure below shows the situation.



We shall assume that flipping edge (a, b) creates a new separating triangle, and show that flipping one of the other edges does not create one. We know that the separating triangle that was created by flipping (a, b) must be (d, c, e), and that (d, c) and (c, e) are edges of the original graph. Triangles (a, d, c) and (b, d, c) must be empty, otherwise (a, b, c) would not be an innermost separating triangle. We now know that the structure of the graph near triangle (a, b, c) is:



Since the graph has at least six vertices, we know that there must be another vertex f outside of triangle (e, b, c) such that (b, f, c) is an empty triangle. Now it is clear that flipping edge (b, c) cannot create a separating triangle.

This completes our construction of a Hamiltonian circuit.

6.2. Transforming the Hamiltonian circuit into a wheel with hubs 1 and n.

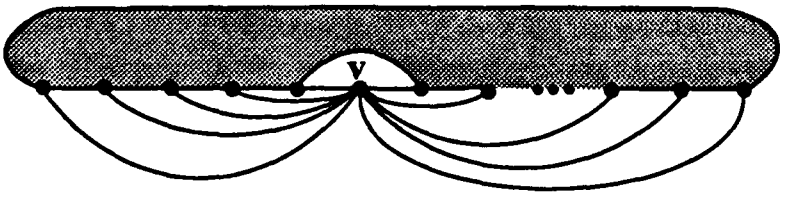
Given that there is a Hamiltonian circuit, we can regard the graph as consisting of a cycle and two triangulations of an n-gon, one on each side of the cycle. By definition, a triangulation of a polygon has no interior vertices. A *coning* triangulation of a polygon is one in which all of the interior edges of the polygon are incident to the same vertex. We shall make use of several facts about diagonal flips in triangulations of a polygon [6]:

Fact 1: Any triangulation of an n-gon can be transformed into the coning triangulation with all edges incident on a vertex v by making at most $n-2$ diagonal flips, each of which increases the degree of v by one.

Fact 2: Any triangulation of an n-gon can be transformed into any other in at most $2n-4$ diagonal flips.

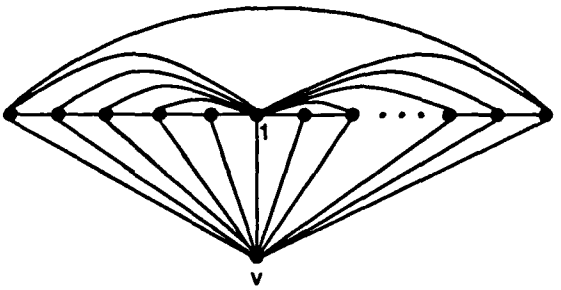
Fact 3: In any triangulation of an n-gon, there is a vertex v such that v is incident to only two edges, and those are the boundary edges that connect v to its two neighbors around the polygon.

Call the two triangulations of the n-gon that comprise the current version of G the *top* triangulation and the *bottom* triangulation. Let v be a vertex such that Fact 3 holds for v in the top triangulation. Now we can apply Fact 1 to vertex v in the bottom triangulation, to transform that triangulation into a coning triangulation to vertex v. This process will never introduce a multiple edge, because all the new edges added to the bottom side of the triangulation are incident to vertex v, which has no edges on the top side. The situation is represented by the following picture:

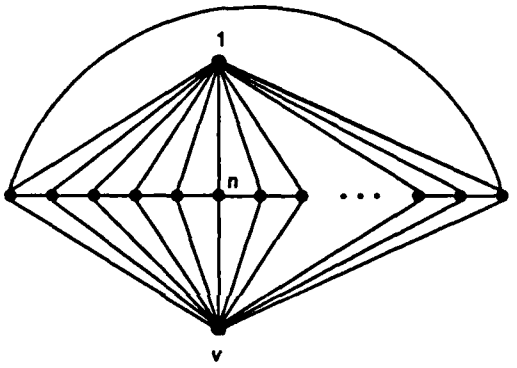


We now change our definition of the top and bottom sides. We view vertex v as belonging to the interior of the bottom side, which is a hub with $n-1$ spokes connecting v to all other vertices. The top side becomes a triangulation of an $(n-1)$-gon. At least one of vertices 1 or n is on this $(n-1)$-gon. Without loss of generality, assume that 1 is on this $(n-1)$-gon. (If 1 is not on this polygon, then the following construction can be fixed by swapping the roles of n and 1.) Now we transform the triangulation of the $(n-1)$-gon (the top side) into

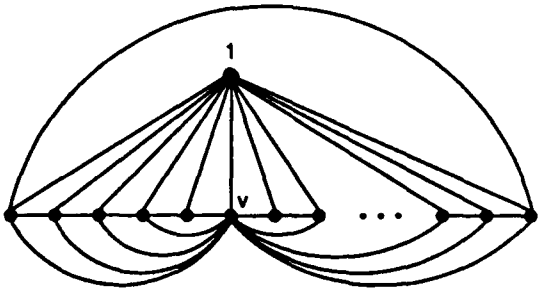
a coning triangulation to vertex 1. The result is shown below.



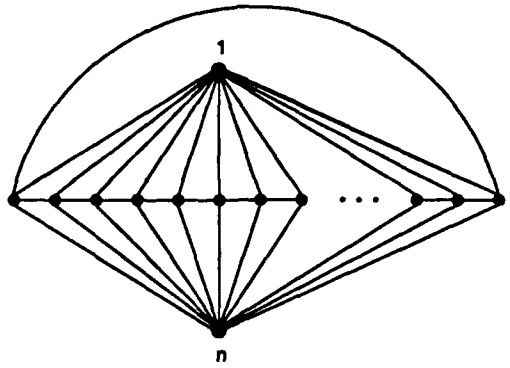
We now flip edge $(v, 1)$ and move 1 into the top side. The result is a wheel with hubs 1 and v, as shown below.



It remains to transform this wheel into one with vertices 1 and n as the hubs. If $v = n$ then we're done, otherwise it only remains to replace v by n. We begin by flipping any edge around the rim of the wheel. The result looks like:



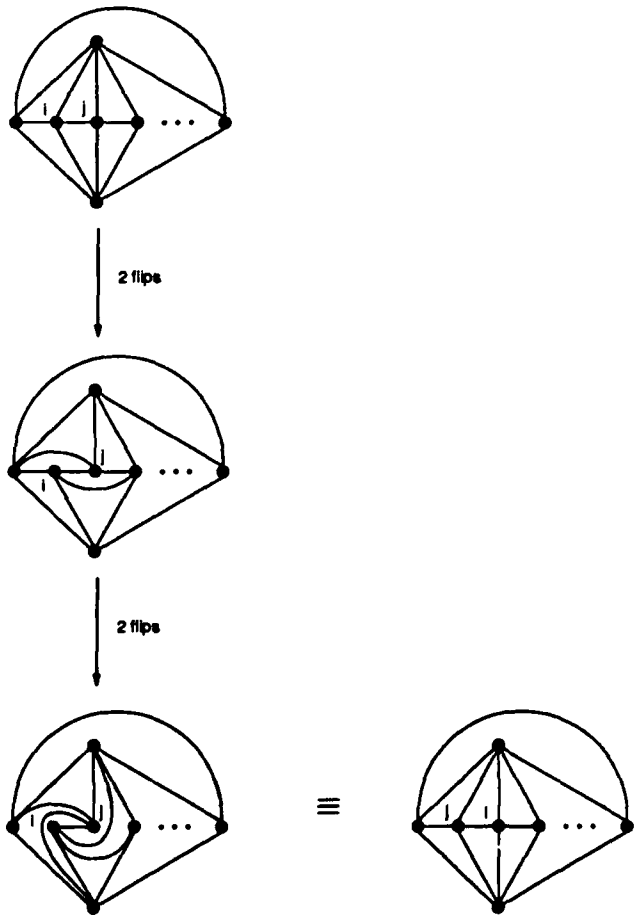
Now we retriangulate the bottom $(n-1)$-gon so that it is a coning to n. Then we flip edge $(n, 1)$, and move n into the bottom half to give the following triangulation:



This construction works without creating multiple edges as long as the rim of the wheel is always at least of size four. This is certainly the case since $n \geq 6$.

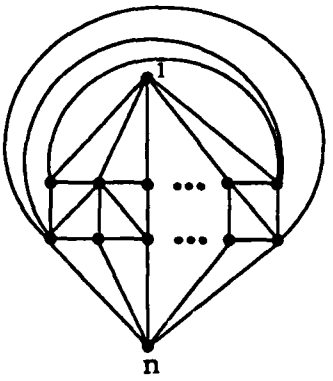
6.3. Sorting the rim of the wheel.

We first give a sequence of four flips that can transpose the order of any pair of adjacent vertices around the rim of the wheel. The following diagram shows this sequence. This will work as long as the number of vertices on the rim is at least four.



If $6 \leq n \leq 15$ we can use repeated transpositions to sort the wheel. From now on we assume that $n \geq 16$.

A *double wheel* is a wheel with two rims, as shown below:

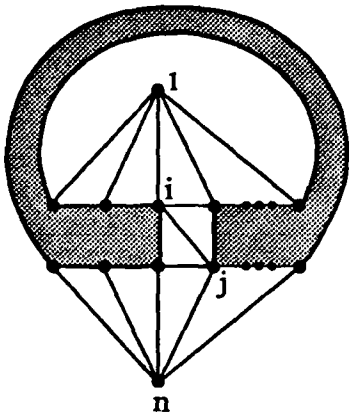


The number of vertices in the *top rim* differs from the number in the *bottom rim* by at most one. Furthermore, all the edges in the region bounded by the two rims cross from one rim to the other.

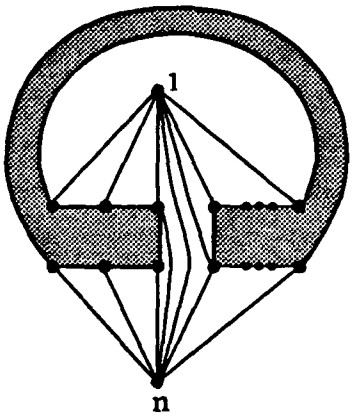
We shall now show how to use $O(n)$ diagonal flips to transform a double wheel into a single wheel. We call this transformation a *merge step*. The merge allows us to form any ordering of the vertices around the rim of the wheel subject to the constraint that the order is consistent with that defined by the orderings on the two rims of the double wheel. That is, if we traverse the rim of the wheel in clockwise order (from the point of view of, say, vertex 1), then the traversal will encounter all the vertices that came from the bottom rim (top rim) of the double wheel in the same cyclic order in which they occurred in the bottom rim (top rim) of the double wheel. A more intuitive way to think of this process is to imagine two decks of cards (the double wheel) which are shuffled into one (the rim of the wheel). This is also analogous to the way a merge sorting algorithm combines two sorted subfiles into a sorted file.

One can also apply the merge step in reverse (an *unmerge*) to split a wheel into a double wheel. A wheel can be sorted by applying a sequence of $\lceil \log_2(n-2) \rceil$ unmerge-merge pairs. (Observe that a merge sort can be implemented using these primitives. Each pass of the sorting algorithm through all of the data corresponds to one of the unmerge-merge pairs.)

It remains to show how to implement the merge step (never introducing multiple edges) in $O(n)$ diagonal flips. An edge (i, j) is called an *amicable edge* if (1) i is on one side of the rim of a double wheel and j is on the other side, (2) the quadrilateral obtained by removing edge (i, j) has one edge on each rim of the wheel and two edges crossing from one side of the rim to the other, and (3) the other vertex of the quadrilateral on the same side of the rim as i is counterclockwise from i (with respect to 1). The following diagram shows an amicable edge (i, j).



In any double wheel there must be an amicable edge. By flipping three edges in the vicinity of an amicable edge we can create an $(n-2)$-gon such that the edges on the outside of the polygon do not connect any pair of vertices of the polygon. When this operation is applied to the diagram above the result is shown below.



It is now the case that we can apply any algorithm for retriangulating the $(n-2)$-gon between the two rims without fear of creating multiple edges.

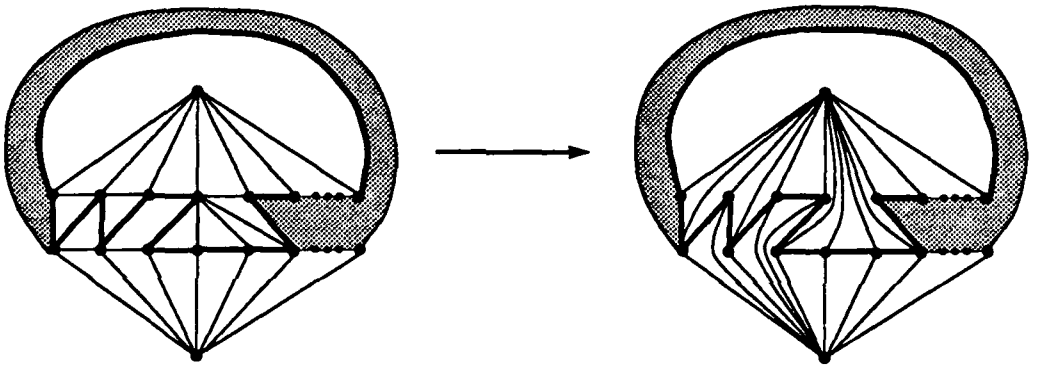
We can apply this technique three times to transform a double wheel with one triangulation between the rims into a double wheel with any other triangulation between the two rims. Let (i,j) be an amicable edge of the initial double wheel. Let (k,l) be an amicable edge of the desired final triangulation. (These pairs of vertices may or may not be disjoint.) Let x and x_c be two neighbors on the top rim of the wheel such that x_c is a counterclockwise neighbor of x, and neither x nor x_c is i, or k, or a counterclockwise neighbor of either i or j. Since the length of the rim is at least 7, there must be such a pair. Define y and y_c similarly on the bottom rim.

To transform the triangulation between the rims from any one to any other, we first cut the double rim at amicable edge (i,j) as shown in the figure above. We then re-triangulate the region between the two rims such that (x,y) is an amicable edge. Then we close up the cut of amicable pair (i,j) and open up the one for amicable edge (x,y). We then

retriangulate the polygon between the rims so that the pair (k, l) becomes an amicable edge. We then close up the cut at amicable edge (x, y), and open up the cut at pair (k, l). Now we triangulate the $(n-2)$-gon as specified by the desired final triangulation between the rims. Closing up the cut at amicable pair (k, l) completes the construction of the desired triangulation between the rims. This process can never introduce any multiple edges, and it uses $O(n)$ diagonal flips.

As the desired triangulation between the rims, we choose any one such that there is an edge joining each pair of vertices that are adjacent on the rim of the desired wheel. It is easy to see that there must be such a triangulation between the rims.

The last step of the process is to convert such a double wheel into a single wheel. The figure below illustrates how this is done. The highlighted edges are those of the rim of the wheel.



This step does at most one flip for each vertex of the rim of the wheel, and completes the merging process. This also completes the proof of the theorem. □

Corollary 1 *Let G_1 and G_2 be two n-vertex numbered plane triangulations (possibly with multiple edges). There exists a sequence of $O(n \log n)$ diagonal flips that transforms G_1 into G_2 in such a way that no flip ever creates a self-loop.*

Proof. An easy case analysis proves the result for $n = 4$. We will show that multiple edges can be eliminated by flipping them one at a time. This takes only a linear number of flips, since each edge is flipped at most once. The corollary result follows by applying Theorem 5 to the graphs obtained by eliminating the multiple edges from G_1 and G_2.

We now prove our claim that if any multiple edge in a plane triangulation is flipped, the number of multiple edges is reduced. Let e_1 and e_2 be a pair of edges between vertices v and w in a plane triangulation. The cycle (e_1, e_2) divides the vertices (except v and w) into two disjoint sets, those on one side of the cycle, and those on the other side. Neither of these two sets can be empty, since every face of the graph is a triangle. If either edge e_1 or e_2 is flipped, it is replaced by an edge that connects a vertex on one side of the cycle to one on

the other side. Since before the flip there were no edges between vertices in these two sets (they are separated by a cycle) the edge created by the flip cannot be a multiple edge. □

References

[1] Culik, K., D. Wood, A note on some tree similarity measures, *Inform. Process. Lett.* 15(1982):39–42.

[2] Ehrig, H., M. Nagl, G. Rozenberg, A. Rosenfeld (Eds.) Graph-grammars and their application to computer science, *Lecture Notes in Computer Science* 291, Springer-Verlag, 1987.

[3] Driscoll, J. R., N. Sarnak, D. D. Sleator, R. E. Tarjan, Making data structures persistent, *J. Computer and System Sci.* 38(1989):86–124.

[4] Knuth, D. E., *The art of computer programming, Vol 1: Fundamental Algorithms*, Addison-Wesley, Reading, MA, 1968.

[5] Knuth, D. E., *The art of computer programming, Vol 3: Sorting and Searching*, Addison-Wesley, Reading, MA, 1973.

[6] Sleator, D. D., R. E. Tarjan, W. P. Thurston, Rotation distance, triangulations, and hyperbolic geometry, *J. Amer. Math. Society* 1(1988):647–681.

[7] Tutte, W. T., A theorem on planar graphs, *Trans. Amer. Math. Soc.* 82(1956):99–116.

[8] Wagner, K., Bemerkungen zum Vierfarbenproblem, *J. Deutschen Math.-verein.* 46(1936):26–32.

[9] Welzl, E., Encoding graph derivations and implications for the theory of graph grammars. *Lecture notes in computer science 172* pp. 503-513, Springer-Verlag, 1984.

[10] Whitney, H., A theorem on graphs, *Ann. of Math.* 32(1931):378–390.